M1 INFO IHM - Introduction à Unity

Hakim Ferrier-Belhaouari

20 février 2026

Les moteurs de jeu vidéo

Inclus des moteurs

- graphique 3D
- graphique 2D
- physique
- audio (2D/3D)
- réseau
- intelligence artificielle (pathfinding, ...)
- animation
- ...

En bref

- C'est compliqué car mêle beaucoup de spécialité.
- Au final, nous avons une boucle perpétuelle de mise à jour de l'image produite et de l'animation d'une *frame*.

Unity

C'est quoi ?

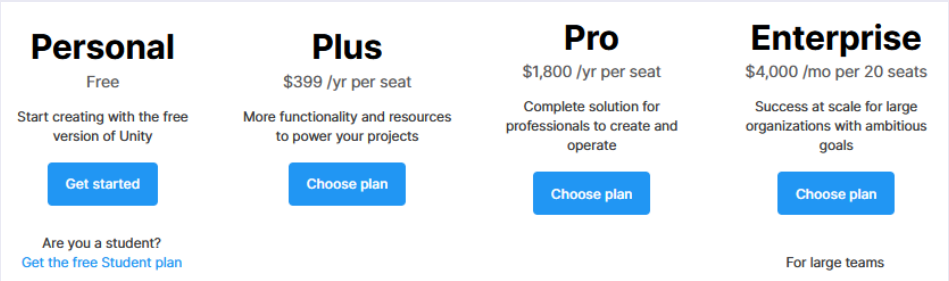
Unity est un **moteur de jeu multiplateformes** pour les jeux vidéos sous la forme d'un **environnement de développement intégré**.

Historique

- 2000 : Création des prémisses d'Unity par 3 danois : volonté de mettre en avant leur expertise.
- 2004 : Fondation de Unity Technologies à San Francisco.
- 2005 : première version de Unity (double licences).
- 2009 : ajout de Direct3D.
- 2010 : création de l'Asset Store (achat d'objet ou autre).
- 2011 : achat de Mecanim : ajout d'animations simples directement dans Unity + ajout de DirectX 11.
- 2013 : ajout d'optimisation des jeux 2D pour répondre à la demande.
- 2015 : intégration de NVIDIA PhysX.
- 2017 : intégration un rendu graphique temps réel.

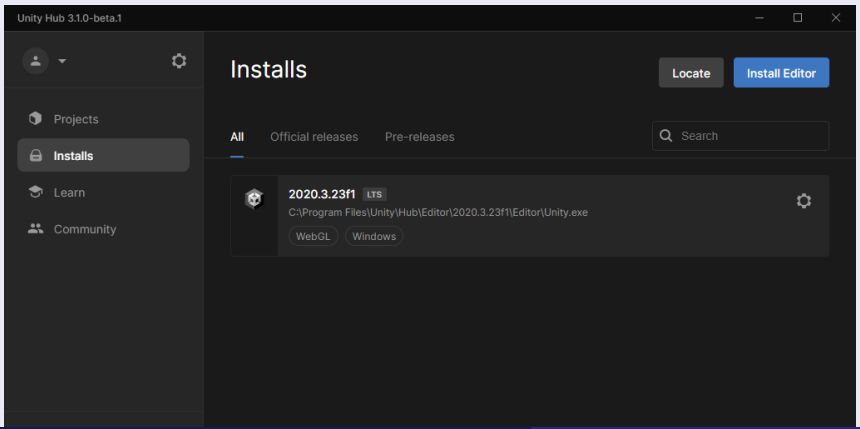
Versions

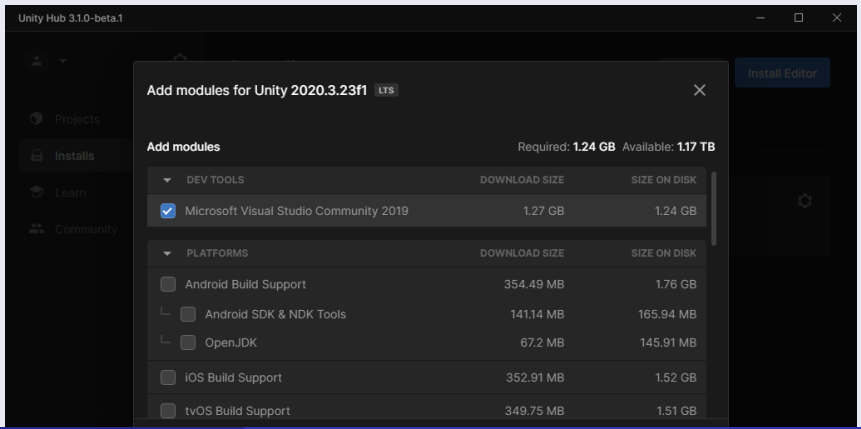
Succès pour des raisons économiques



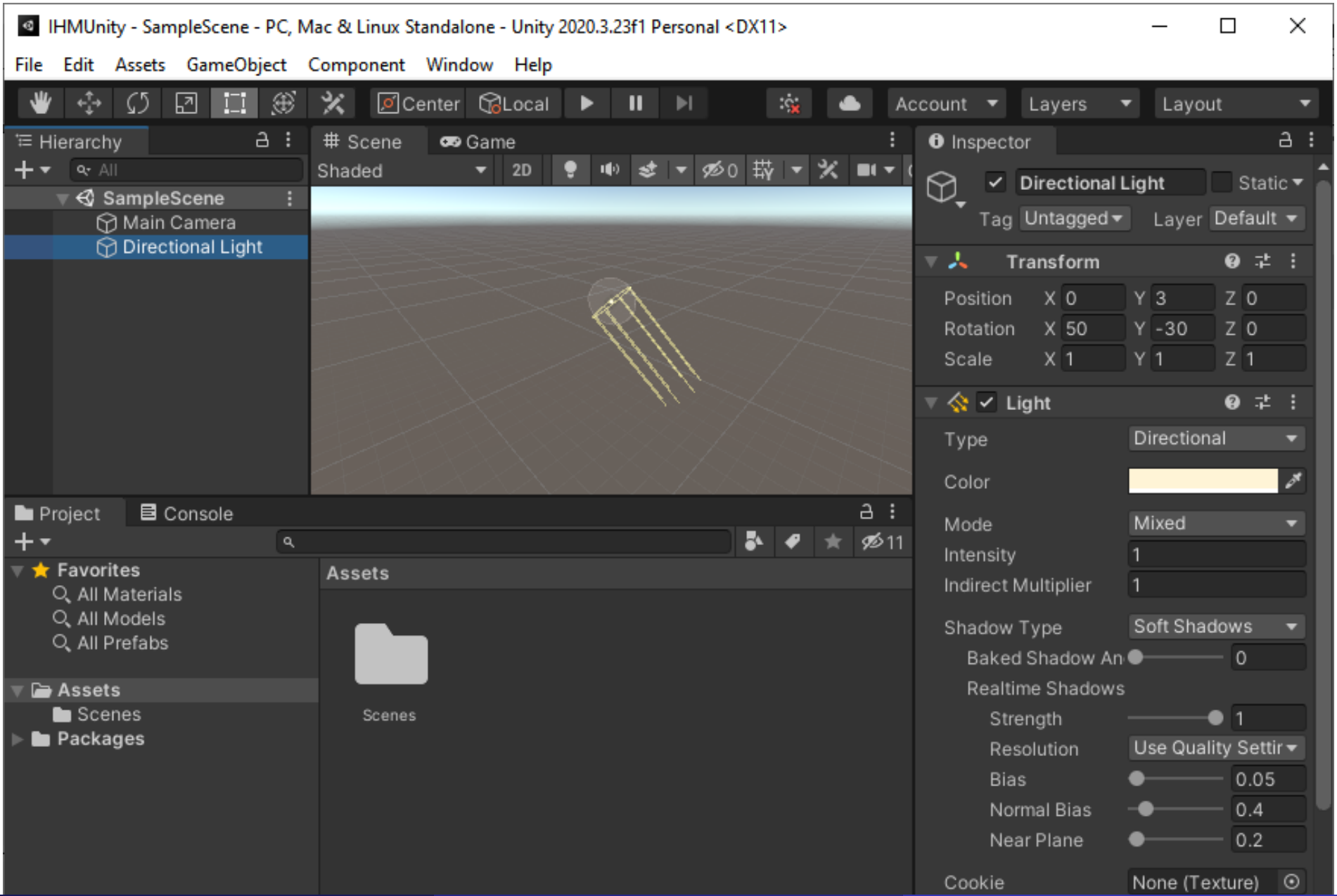
La version personnelle est amplement suffisante pour ce cours, car fonctionne sous les plateformes classiques.

Mécanisme de HUB (version cours : 6000.3.XXX chez vous (on verra en TP))





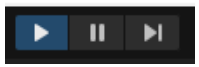
Lancement d'Unity



Lancement d'Unity

Les onglets sont importants et en voici les principaux :

1. Hierarchy : donne une vue sous forme d'arbre de la zone 3D et des objets présents dans votre `scène`(= correspond à un niveau de votre jeu) ou vos scènes.
2. Scene : présente la vue 3D de la scène en cours.
3. Game : vu de votre application quand vous l'exécutez en direct.
4. Inspector : donne l'ensemble des propriétés de votre sélection.
5. Project : permet d'explorer l'entièreté de votre projet (dont les Assets)
6. Console : donne tous les messages de log de Unity.

La barre de lancement 

1. lance l'execution de votre application directement. Et arrête son exécution aussi.
2. met en pause l'exécution (pratique pour déboguer).
3. et le bouton step toujours pour le débogage.

Demo1

Objectifs

- Créez un projet 3D nommé `IHMUnity_Demo1`.
- Regardez les paramètres de votre application (Project Settings > Player)
- Ajoutez des objets primitives (1 cube et 1 sphère) dans la vue 3D.
- Regardez leurs propriétés et repérez les similitudes et différences.
- Lancez l'exécution de la vue.
- Pourquoi rien ne se passe ?
- Modifier la position du cube en direct sans arrêter la simulation.
- Stopper la simulation.
- Que remarquez vous sur les positions ?

La classe GameObject

Definition

La classe **GameObject** est la classe de base de tous les objets présents dans une scène. Leur spécificité est la notion de composant (**Component**).

Exemple : Transform

Le composant Transform gère le position de l'objet dans la scène.

Exemple : MeshFilter

Le composant MeshFilter gère le maillage de l'objet.

Les composants

Les composants représentent le cœur de la programmation en Unity. Ainsi, pour un objet il suffit de lui ajouter le bon composant pour lui modifier/influencer/ajouter un comportement ou une mécanique.
Lorsqu'il n'y a pas de composant adéquat à votre besoin il nous suffit de le programmer !

Demo2

Objectifs

Sur le projet précédent :

- Ajoutez un composant système de particule au cube.
- Amusez vous pour voir les modifications induits en simulation.
- Stoppez tout.
- Ajoutez un objet 3D Quad, plane ou cube.
- Ajoutez un composant vidéo.
- Lancez la simulation pour voir l'animation de la vidéo sur l'objet.
- Stopper la simulation.
- Que remarquez vous sur les positions ?

Les interfaces

Attention !

Il existe plusieurs kits d'interface graphiques en Unity :

Type of UI	UI Toolkit	Unity UI(uGUI)	IMGUI	Notes
Runtime (debug)	✓*	✓	✓	This refers to temporary runtime UI used for debug purposes.
Runtime (in-game)	✓*	✓	Not Recommended	For performance reasons, Unity does not recommend IMGUI for in-game runtime UI.
Unity Editor	✓	✗	✓	You cannot use Unity UI to make UI for the Unity Editor.

* *Requires the UI Toolkit package, currently in preview.*

Dans ce cours, nous ferons que du `Unity UI` qui suit les mécaniques des `GameObject`. Cependant, il est impossible d'étendre l'éditeur.

La zone d'interface se compose

Lorsque vous réalisez une interface il est OBLIGATOIRE d'avoir les éléments suivants (Unity peut vous les créer si besoin) :

- Canvas : la zone de dessin qui est la base de toute interface (cf démo)
- EventSystem : un objet faisant le pont avec les entrées (clavier, souris, ...).

Les interfaces

Dans ce cours

Nous construirons nos widgets graphiques par composition d'élements existant et de script pour compenser des manques.

Spécificité des widgets graphiques

Les widgets graphiques n'ont pas de `Transform` qui gère la position dans la scène, mais un `RectTransform` qui s'occupe du placement dans le canevas. Le paramétrage est différent.

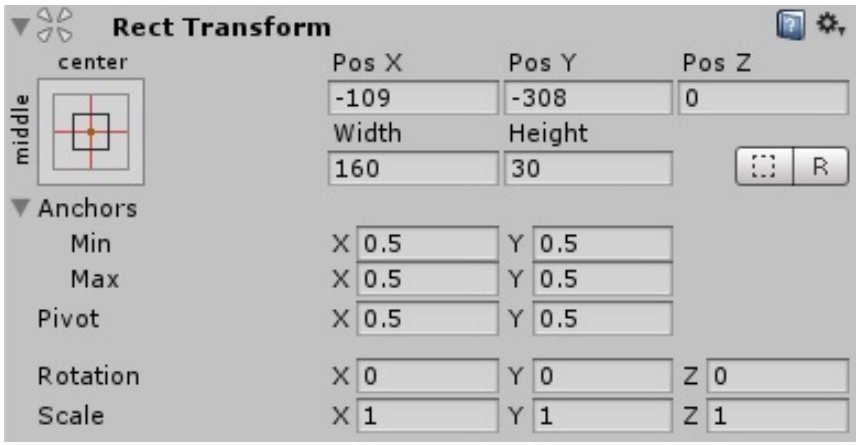
La mise en page

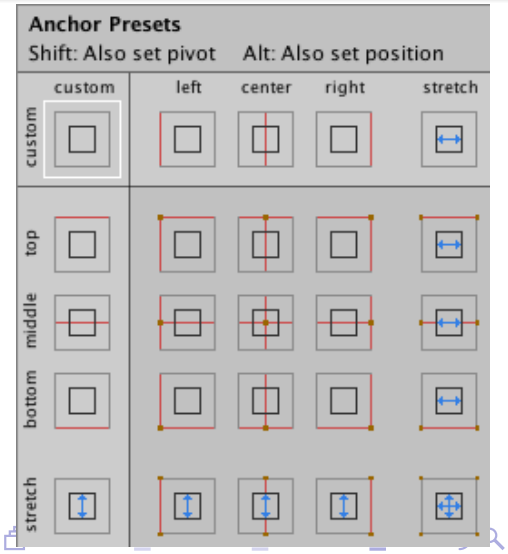
Attention !

Le canevas a toujours une taille précise (dans les paramètres de l'application) et mise à l'échelle. Elle sera donc toujours comme vous la voyez dans l'éditeur !
La mise en page des widgets n'est pas aussi intuitive que les autres outils, en raison de la consommation des replacements.

Les ancres

Tous les widgets ont des ancres qui permettent de fixer leur position au sein de leur parent.





La notion de Préfab

Definition

Un préfab est un objet qui peut être réutilisé à l'infini dans votre projet. Il est composé d'un GameObject et de tous ses composants.

Pourquoi ?

Parce que cela permet de faire du prototypage rapide et de la réutilisation d'éléments dans votre projet.

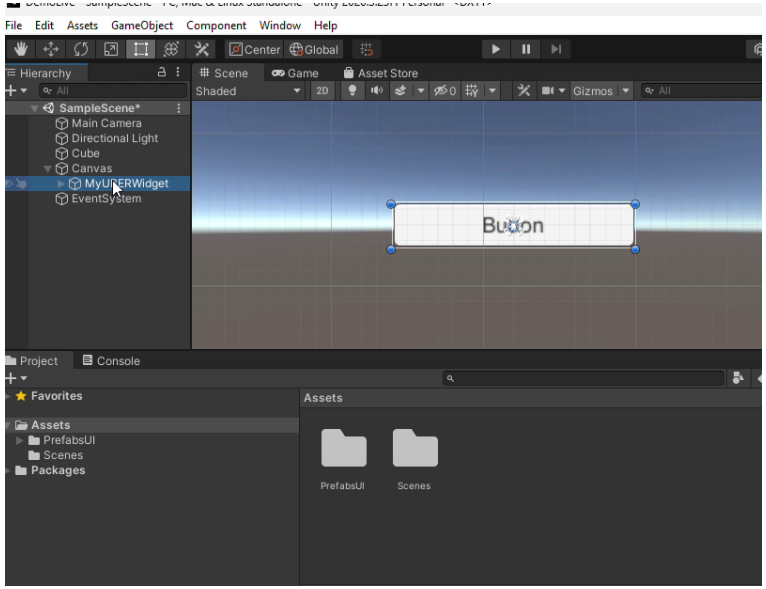
Comment ?

Pour créer un préfab, il suffit de faire glisser un objet de la hiérarchie vers le projet. Vous pouvez ensuite faire glisser ce préfab dans votre scène pour l'instancier.

(cf. démo - prochaine page)

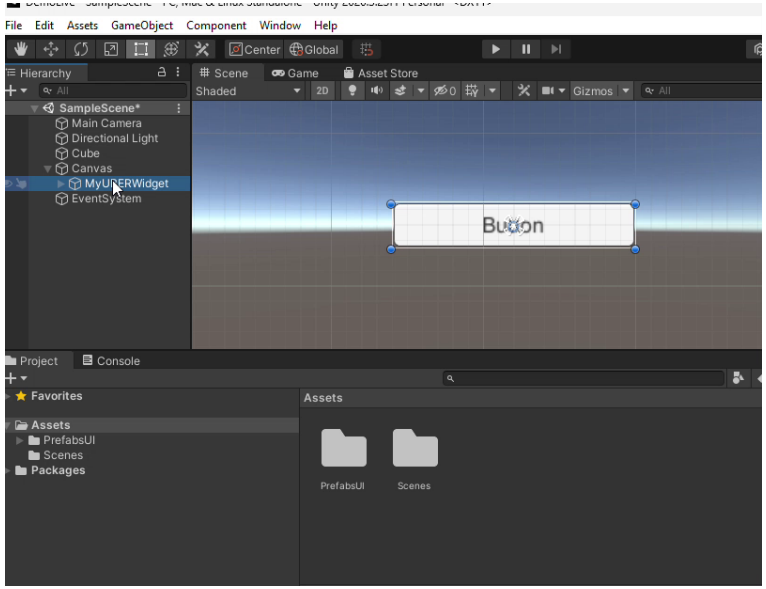
Démonstration : la création d'un préfab

Étape 1

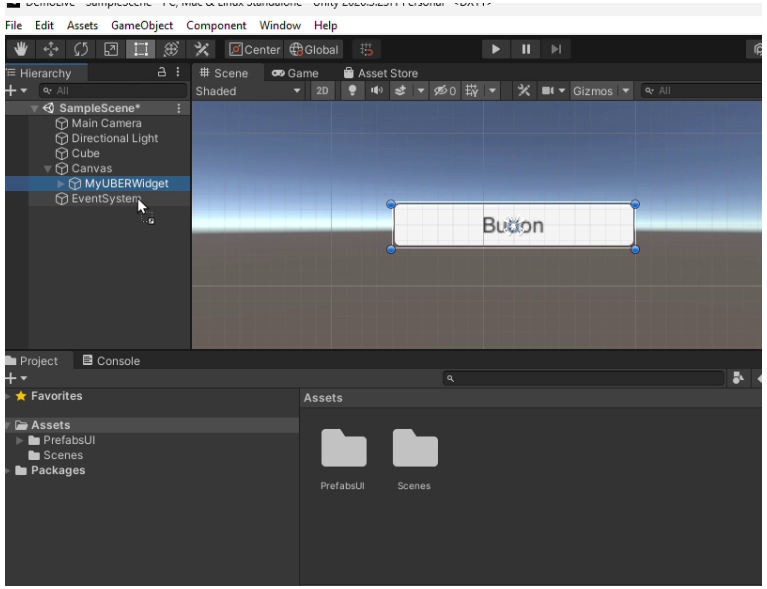


Démonstration : la création d'un préfab

Étape 1

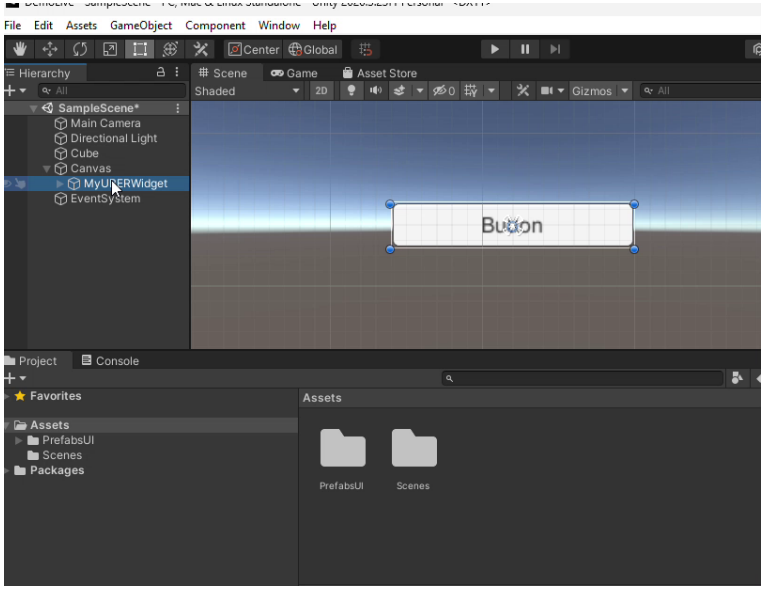


Étape 2

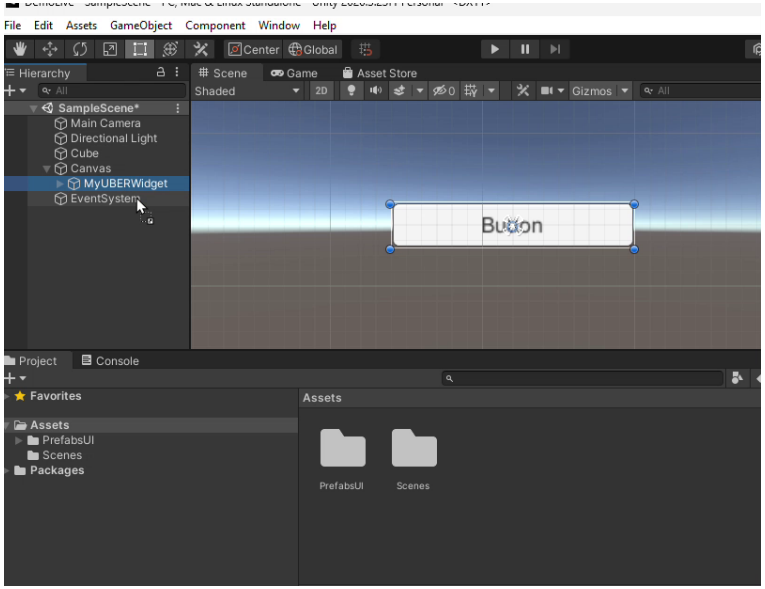


Démonstration : la création d'un préfab

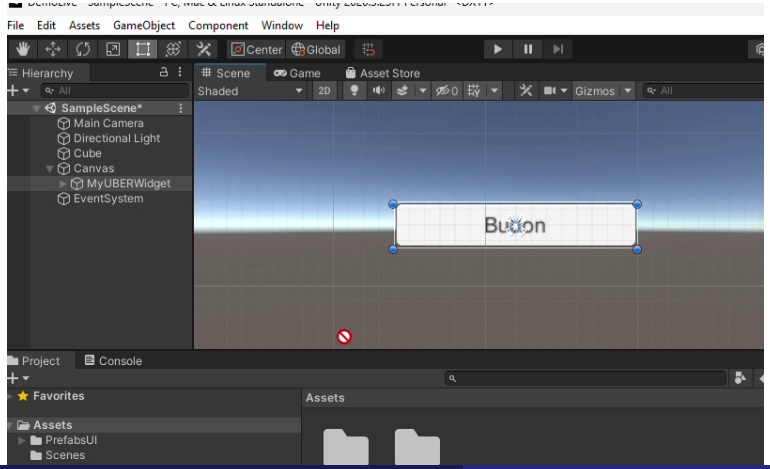
Étape 1



Étape 2

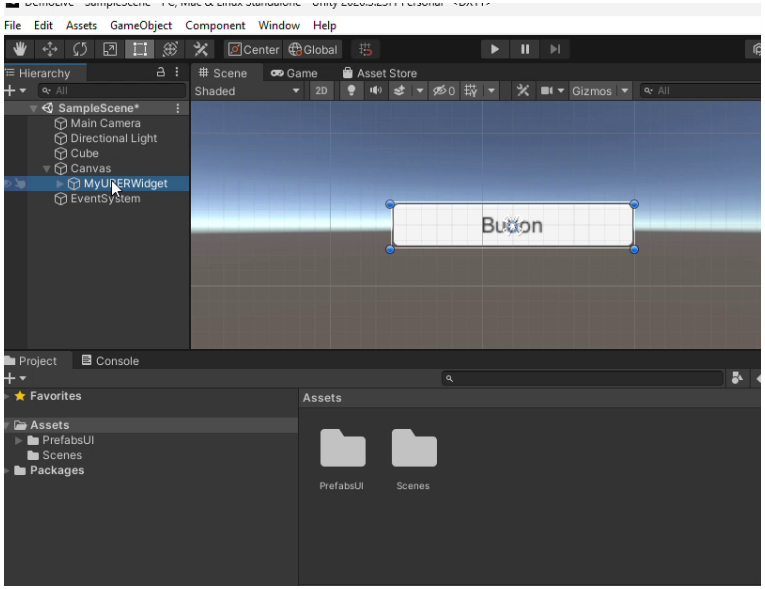


Étape 3

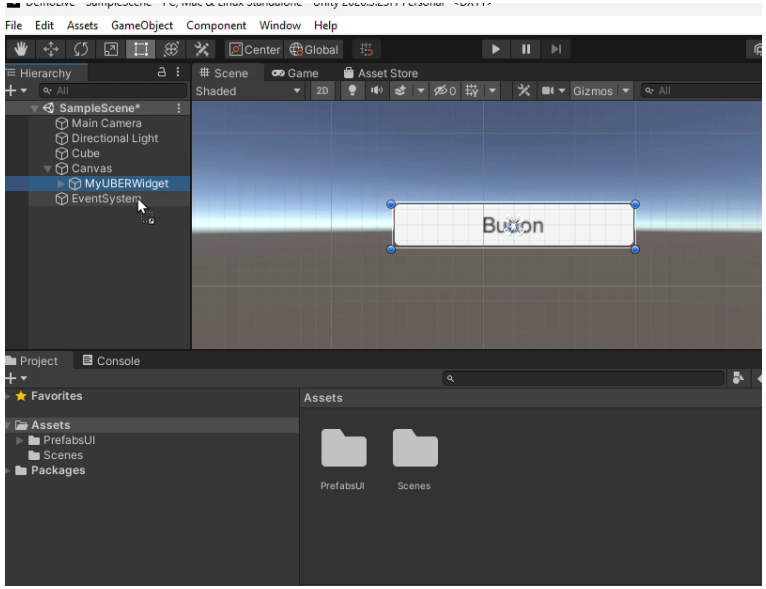


Démonstration : la création d'un préfab

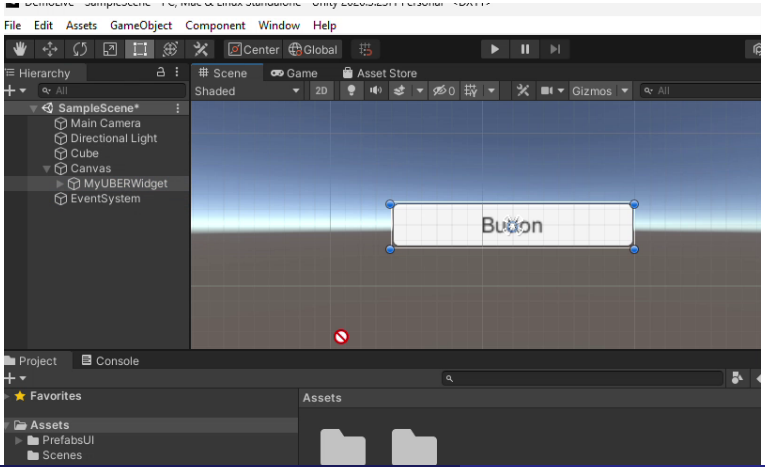
Étape 1



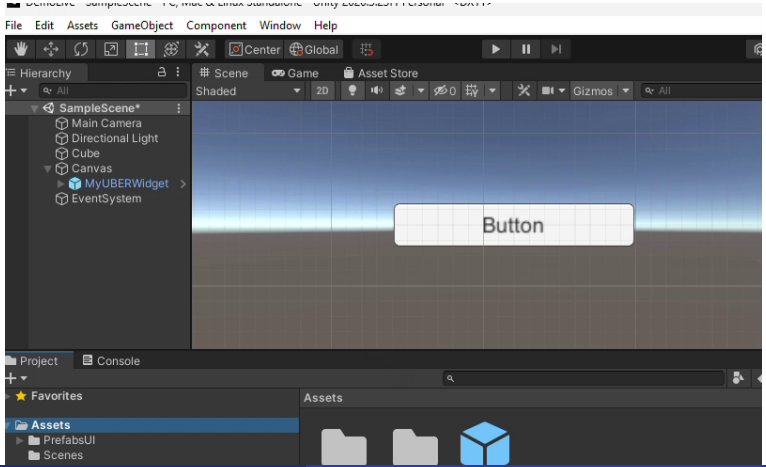
Étape 2



Étape 3

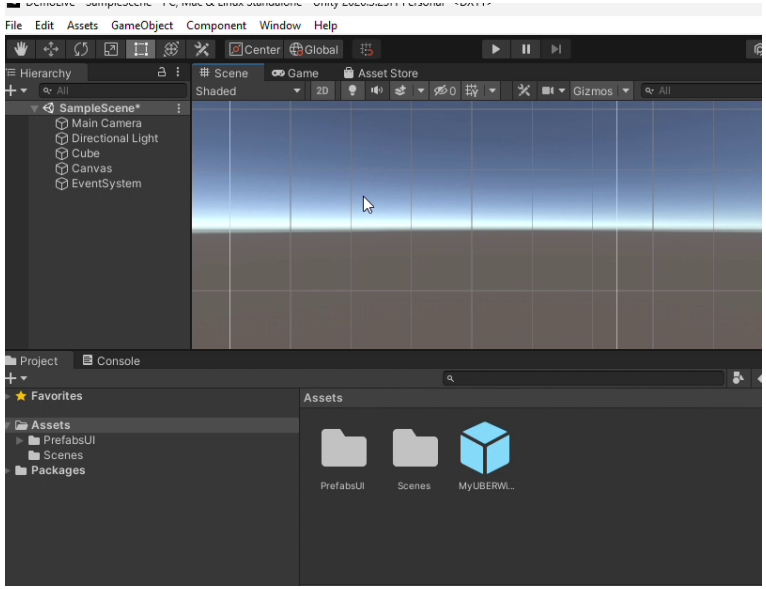


Étape 4



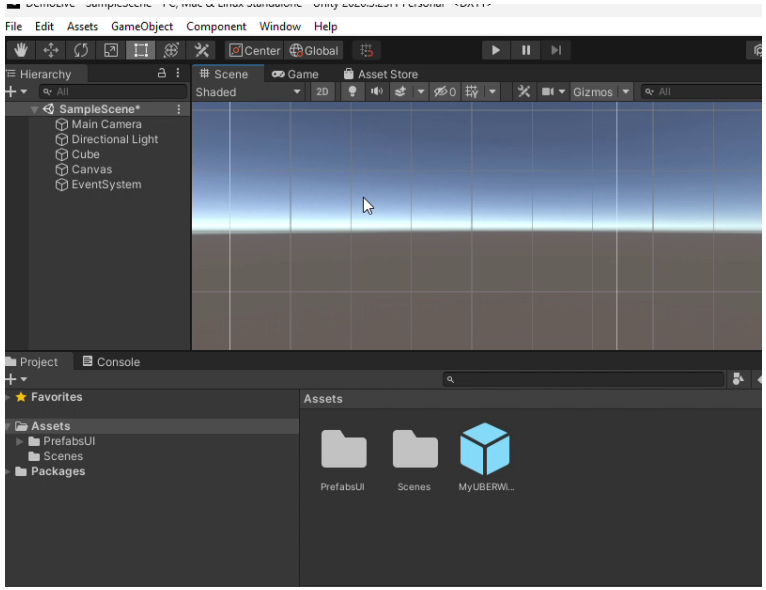
Démonstration : l'utilisation d'un préfab

Étape 1

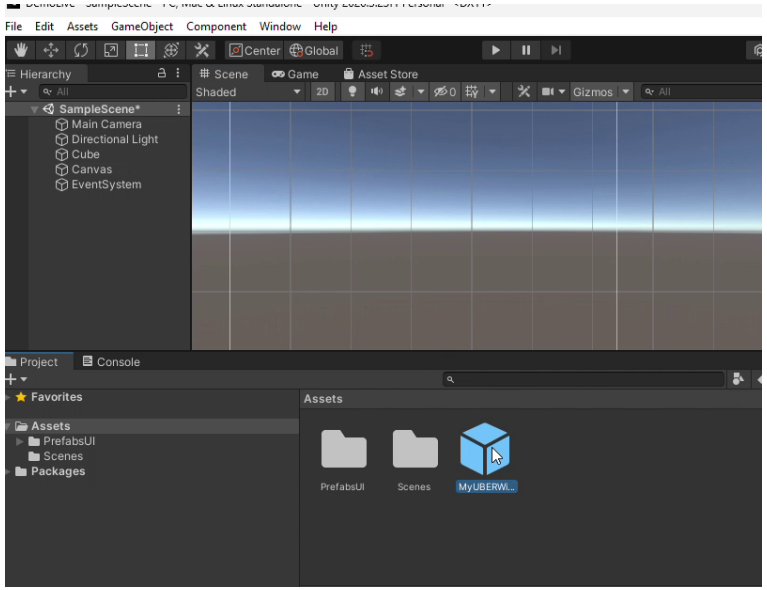


Démonstration : l'utilisation d'un préfab

Étape 1

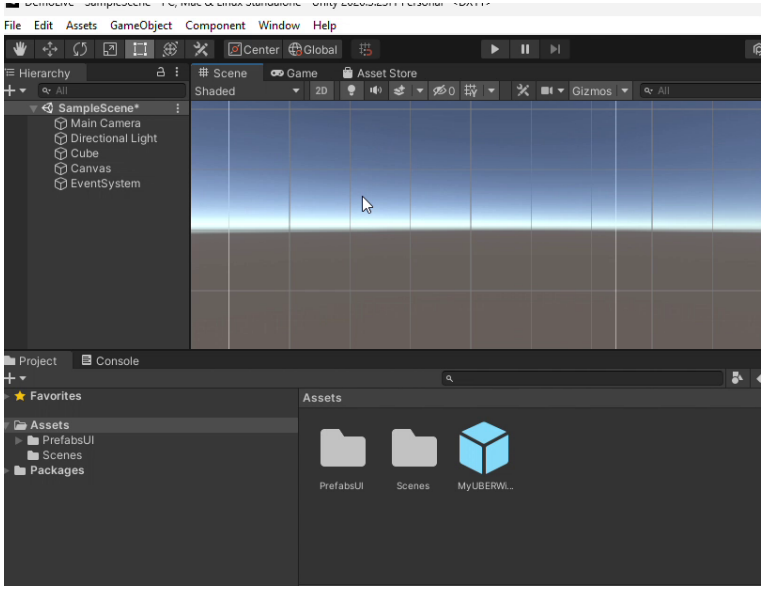


Étape 2

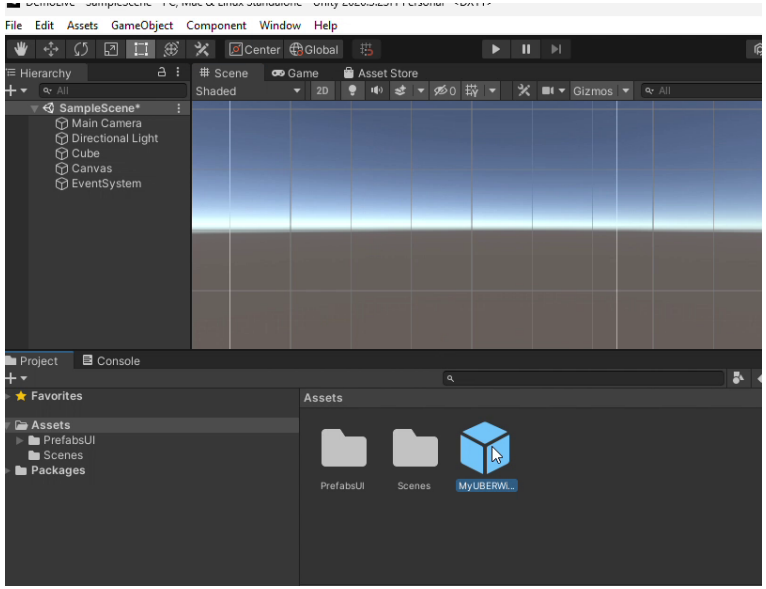


Démonstration : l'utilisation d'un préfab

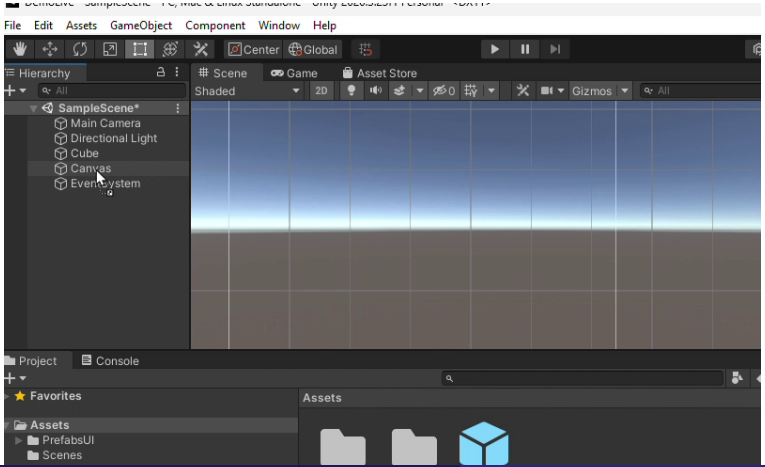
Étape 1



Étape 2

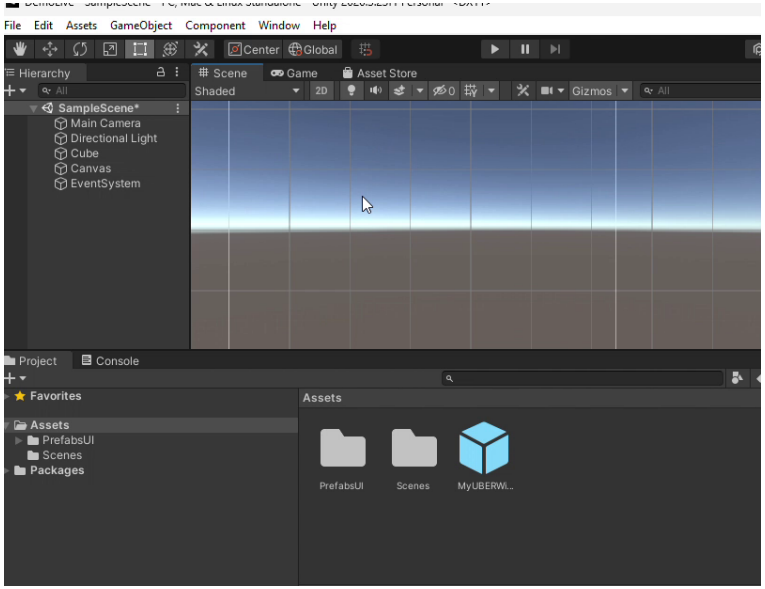


Étape 3

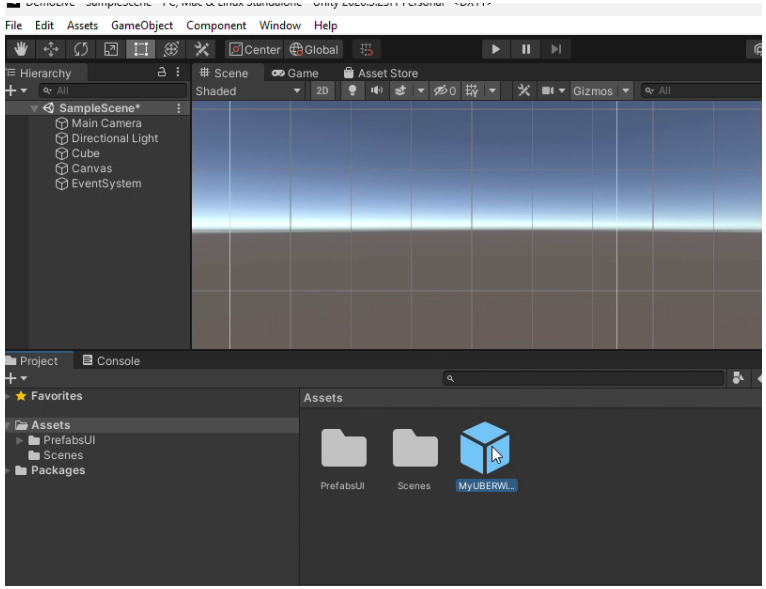


Démonstration : l'utilisation d'un préfab

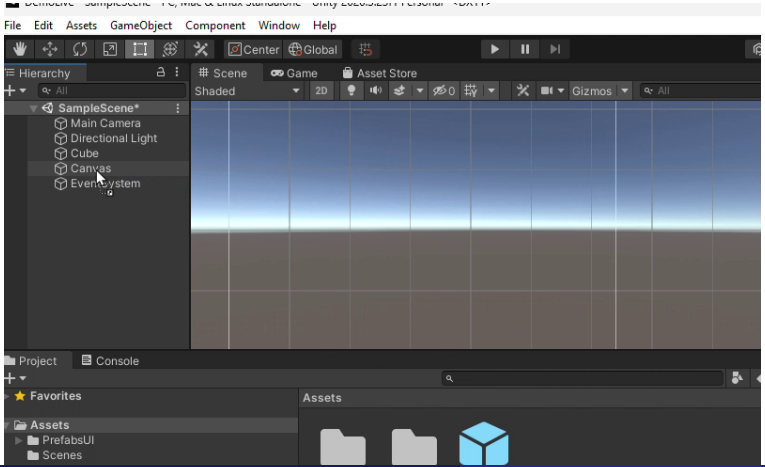
Étape 1



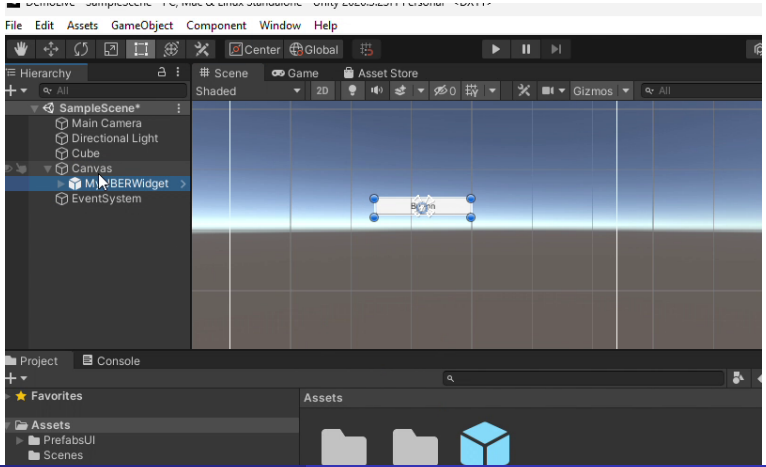
Étape 2



Étape 3



Étape 4



Exportation d'un *widget* graphique

Les *unitypackage*

Un *unitypackage* est un format d'archive utilisé par Unity pour stocker et partager des ressources, des scripts, des scènes, des préfabs, et d'autres éléments de projet. Il permet de regrouper plusieurs fichiers et dossiers en une seule archive compressée, facilitant ainsi le transfert et l'importation de ces éléments dans d'autres projets Unity.

Attention !

- L'exportation d'un *unitypackage* est un processus simple qui permet de partager facilement des éléments de projet entre différents projets Unity.
- Votre soumission finale devra contenir quelques *unitypackage* bien réalisés pour que nous puissions les réutiliser pour la correction.
- Mettez bien les dépendances nécessaires et uniquement ceux nécessaires pour éviter les problèmes d'importation.
- La notation prendra en compte le bon découpage de vos interfaces.

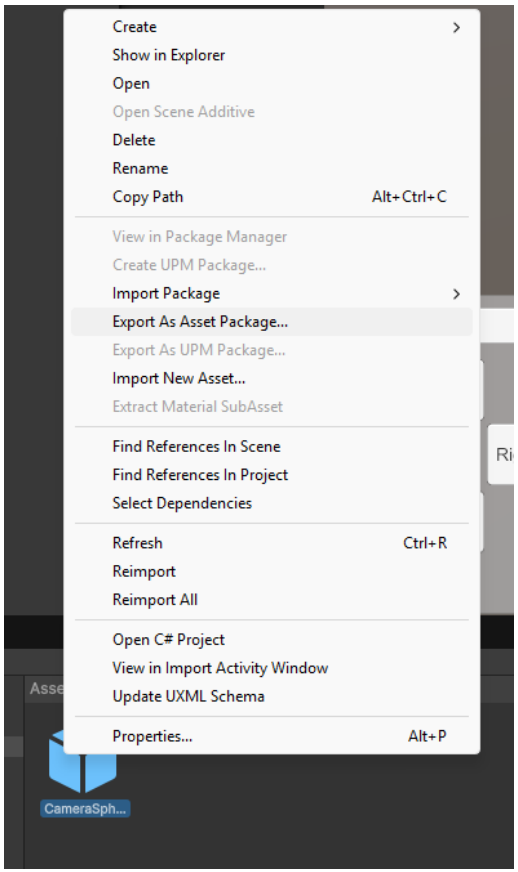
(cf. démo - prochaine page)

Démonstration : l'exportation d'un *widget* graphique

Processus

- Sélectionnez le Préfab ou les éléments que vous souhaitez exporter.
- Faites un clic droit et choisissez "Export Package".

Étape 1

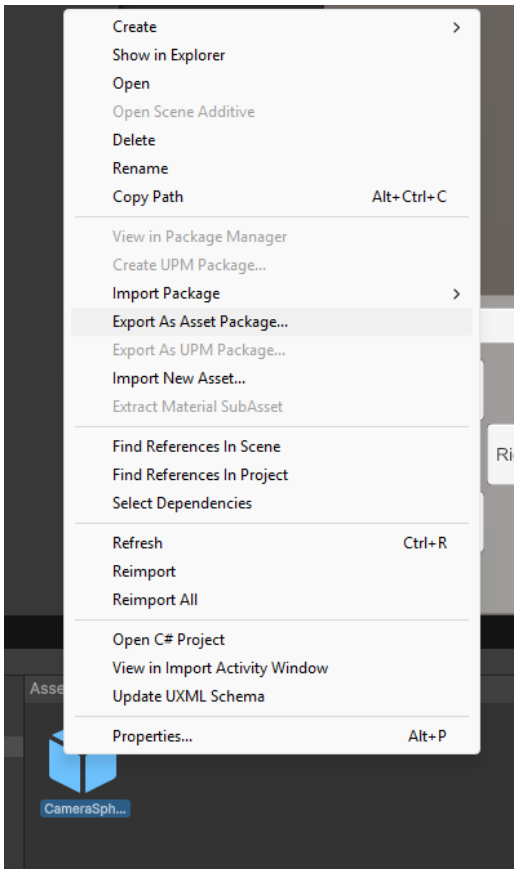


Démonstration : l'exportation d'un *widget* graphique

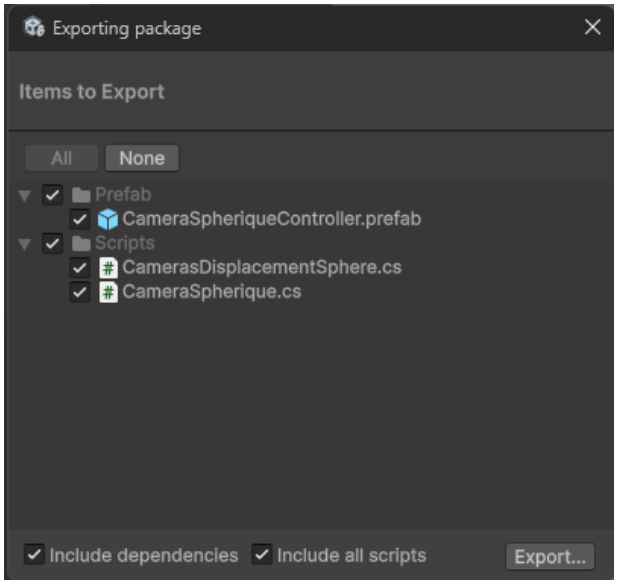
Processus

- Sélectionnez le Préfab ou les éléments que vous souhaitez exporter.
- Faites un clic droit et choisissez "Export Package".

Étape 1



Étape 2

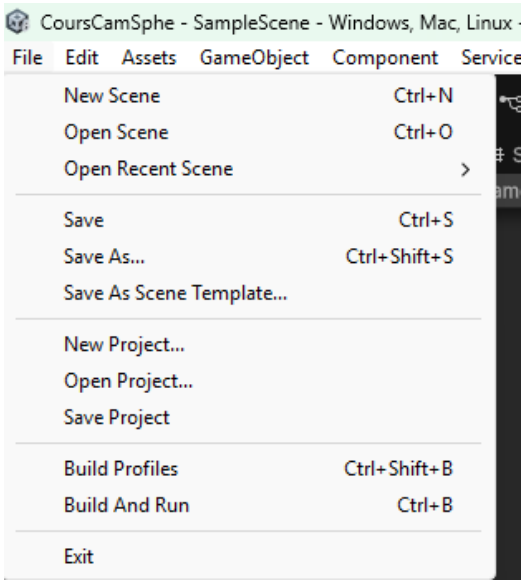


Construction de votre exécutable

Processus de construction

- Allez dans "File" > "Build Settings".
- Sélectionnez la plateforme cible (PC, Mac, Linux, Android, iOS, etc.).
- Configurez les paramètres de construction selon vos besoins.
- Cliquez sur "Build" pour générer l'exécutable dans un répertoire vide.

Étape 1

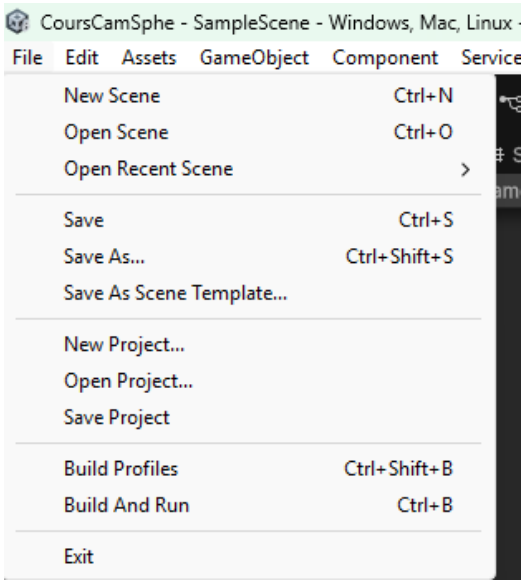


Construction de votre exécutable

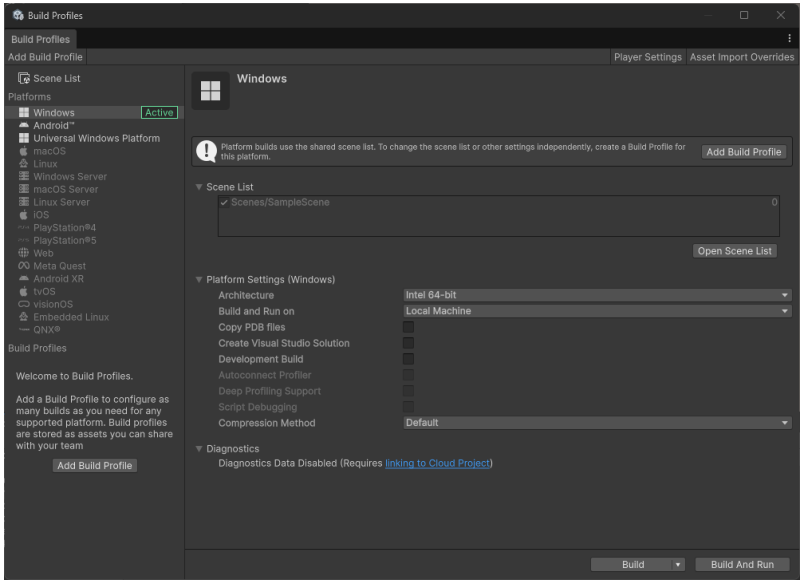
Processus de construction

- Allez dans "File" > "Build Settings".
- Sélectionnez la plateforme cible (PC, Mac, Linux, Android, iOS, etc.).
- Configurez les paramètres de construction selon vos besoins.
- Cliquez sur "Build" pour générer l'exécutable dans un répertoire vide.

Étape 1



Étape 2

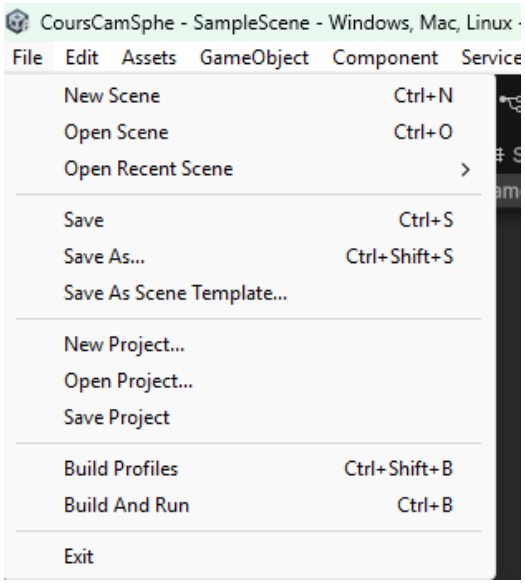


Construction de votre exécutable

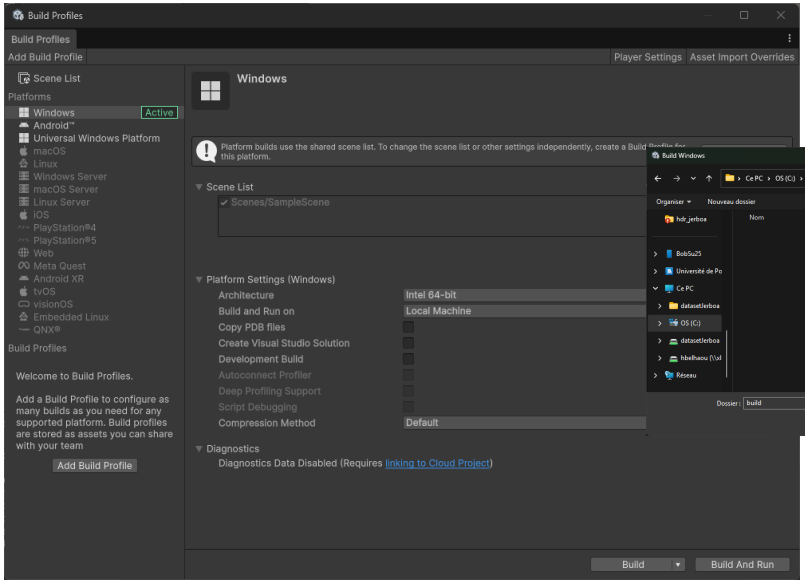
Processus de construction

- Allez dans "File" > "Build Settings".
- Sélectionnez la plateforme cible (PC, Mac, Linux, Android, iOS, etc.).
- Configurez les paramètres de construction selon vos besoins.
- Cliquez sur "Build" pour générer l'exécutable dans un répertoire vide.

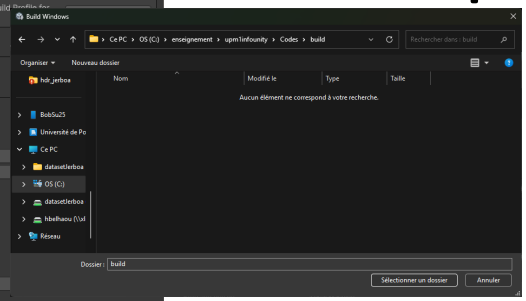
Étape 1



Étape 2



Étape 3



Demo

Objectifs : un widget de contrôle de caméra

- Créez une nouvelle scène.
- Ajoutez un canevas.
- Ajoutez un panel dans le canevas.
- Ajoutez 4 boutons dans le panel.
- Amusez vous à les positionner et à les redimensionner.

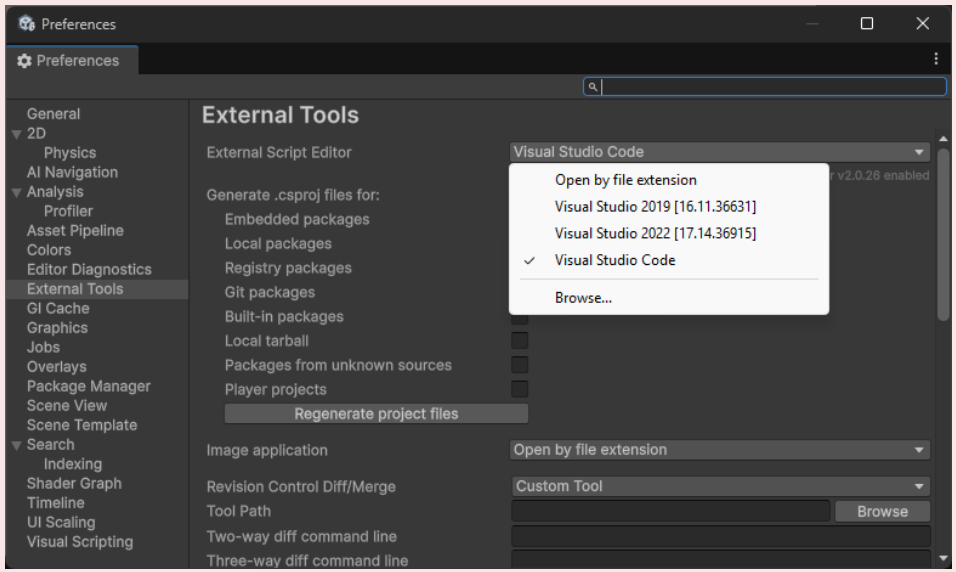
Programmation avec Unity

Langage de programmation

Unity utilise principalement le langage de programmation C#. Ce langage est puissant et très proche de Java.

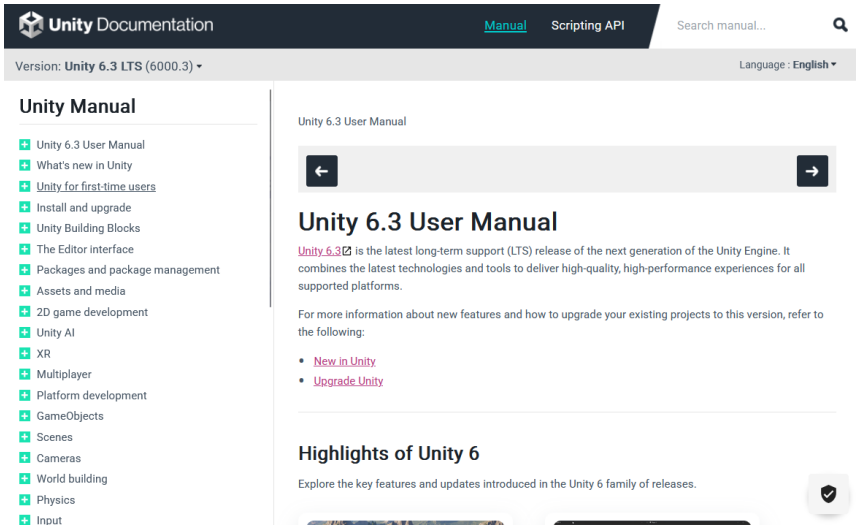
Environnement de développement

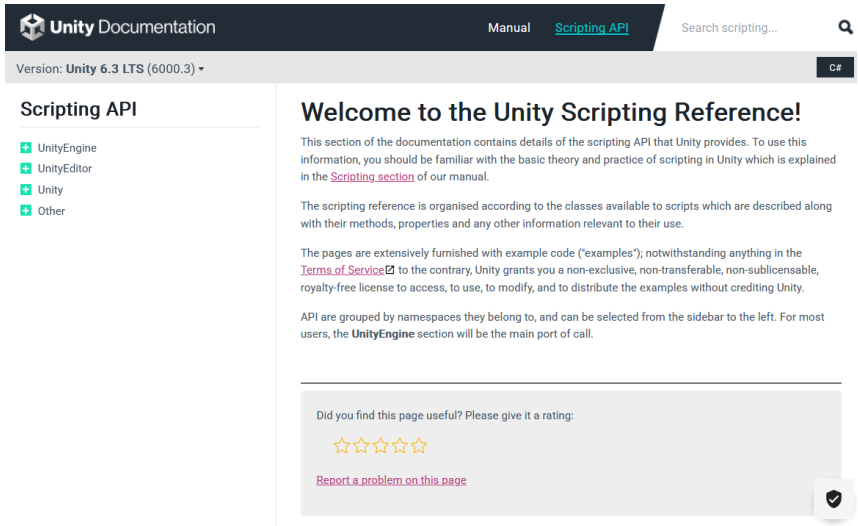
- Choix de l'IDE dans les paramètres d'Unity.
- Sinon risque de ne pas avoir d'autocomplétion ou de fonctionnalités avancées.



La documentation Unity

- Disponible en ligne : `https://docs.unity3d.com/`.
- Selon moi la meilleure documentation par rapport aux autres moteurs de jeu.
- Attention, deux parties distinctes : le manuel d'utilisation et la référence technique (Scripting API).
- 2 espaces de noms distincts : UnityEngine (fonctionnalités à l'exécution) et UnityEditor (fonctionnalités dans le mode éditeur).





Composant script

Composant script

Un script s'ajoute comme un composant à un objet de la scène. Il hérite de la classe MonoBehaviour.

Le cycle de vie d'un script

Plus complexe que sur Android :
https://docs.unity3d.com/Manual/ExecutionOrder.html

- `Awake()` : appelée lorsque le script est chargé pour la première fois.
- `Start()` : appelée avant la première frame de mise à jour, après que tous les objets ont été initialisés.
- `Update()` : appelée une fois par frame, utilisée pour la logique de jeu et les interactions.
- `FixedUpdate()` : appelée à intervalles fixes, utilisée pour la physique.
- `OnDestroy()` : appelée lorsque l'objet est détruit.

Programmation : attributs et états

Variables publiques dans l'inspecteur

Les variables publiques d'un script sont automatiquement affichées dans l'inspecteur Unity, permettant de modifier les valeurs directement sans recompiler.

Exemple simple

- `public int nombre = 10;` — affiche un entier modifiable
- `public bool actif = true;` — affiche une case à cocher
- `public string nom = "objet";` — affiche un champ texte

Bonnes pratiques

Évitez d'exposer des variables publiques critiques. Utilisez plutôt `[SerializeField]` pour les variables privées que vous voulez éditer.

Attributs de contrôle : Range et Clamp

`[Range(min, max)]`

Limite une valeur numérique entre deux bornes avec un curseur dans l'inspecteur.

Exemple

```
[Range(0f, 10f)] public float vitesse = 5f;
```

Affiche un curseur entre 0 et 10 pour contrôler facilement la vitesse.

`[Min(valeur)]` et `[Max(valeur)]`

Impose une valeur minimale ou maximale sans curseur.

Exemple

`[Min(0f)] public float duree = 2f;` — empêche les valeurs négatives

Attributs d'organisation

[Header("Titre")]

Ajoute un en-tête visuel pour organiser les variables dans l'inspecteur.

Exemple

```
[Header("Paramètres de mouvement")] public float vitesse = 5f;
```

[Space(hauteur)]

Ajoute un espacement vertical dans l'inspecteur pour améliorer la lisibilité.

[Tooltip("description")]

Affiche une bulle d'aide au survol de la variable dans l'inspecteur.

Exemple

```
[Tooltip("Vitesse de rotation en degrés par seconde")] public
float vrotation = 45f;
```

Attributs de contrôle avancés

`[TextArea(minLignes, maxLignes)]`

Affiche un champ de texte sur plusieurs lignes.

`[Multiline(nbLignes)]`

Alternative simple pour du texte multiligne.

`[ColorUsage(showAlpha, hdr)]`

Personnalise le sélecteur de couleur (avec ou sans canal alpha, support HDR).

Exemple

```
[ColorUsage(true, true)] public Color couleur = Color.white;
```

Attributs de visibilité

[SerializeField]

Expose une variable **privée** à l'inspecteur (bonne pratique pour l'encapsulation).

[HideInInspector]

Masque une variable **publique** de l'inspecteur.

Exemple complet

```
[SerializeField] [Range(0f, 100f)] [Tooltip("Santé du personnage")] private float sante = 50f;
```

Variable privée, modifiable dans l'inspecteur, limitée entre 0 et 100, avec une description.

Combinaison d'attributs

Vous pouvez combiner plusieurs attributs sur une même variable pour plus de contrôle !

La classe GameObject

Importance fondamentale

GameObject est la **classe de base** de tous les objets dans une scène Unity. C'est le conteneur principal qui regroupe des composants pour créer des entités jouables.

Attributs importants

- `name` : identifie l'objet dans la hiérarchie
- `active` : booléen indiquant si l'objet est actif
- `transform` : référence au composant Transform (position, rotation, échelle)
- `tag` : étiquette pour identifier rapidement des objets
- `layer` : couche de rendu pour les caméras et collisions

Fonctions importantes

- `SetActive(bool)` : active ou désactive l'objet
- `Destroy()` : supprime l'objet de la scène

La classe Transform

Rôle central

Transform gère la **position**, la **rotation** et l'**échelle** d'un objet dans l'espace 3D. Tout GameObject possède obligatoirement un Transform.

Attributs essentiels

- `position` : vecteur Vector3 en coordonnées mondiales
- `localPosition` : position relative au parent
- `rotation` : rotation en quaternion
- `eulerAngles` : rotation en degrés (x, y, z)
- `scale` : échelle sur les 3 axes
- `parent` : référence au Transform parent

Fonctions courantes

- `Translate(Vector3)` : déplace l'objet
- `Rotate(Vector3)` : fait tourner l'objet
- `LookAt(Vector3)` : oriente l'objet vers une position

Les composants physiques

Rigidbody

Ajoute des propriétés physiques à un objet (gravité, collisions, forces).

- `velocity` : vitesse actuelle
- `mass` : masse de l'objet
- `AddForce()` : applique une force
- `SetVelocity()` : modifie la vélocité

Colliders

Définissent les zones de collision (BoxCollider, SphereCollider, MeshCollider, etc.).

- `isTrigger` : rend le collider sans collision physique
- `OnCollisionEnter()` : appelée au contact avec un autre objet
- `OnTriggerEnter()` : appelée quand un objet entre dans la zone

Les composants de rendu

MeshRenderer et MeshFilter

- MeshFilter : contient la géométrie (le maillage) de l'objet
- MeshRenderer : affiche le maillage avec des matériaux et des textures

Camera

Capture la scène et la projette sur l'écran.

- `main` : accès à la caméra principale
- `ScreenPointToRay()` : crée un rayon depuis l'écran
- `WorldToScreenPoint()` : convertit positions 3D en 2D écran

Light

Éclaire la scène (directional, point, spot).

- `intensity` : intensité lumineuse
- `range` : portée (pour point et spot)
- `color` : couleur de la lumière

Les classes d'entrée

Input (ancien système)

Gère le clavier, la souris et les manettes.

- `GetKey(KeyCode)` : touche actuellement appuyée
- `GetKeyDown()` : première frame d'appui
- `GetMouseButton()` : bouton souris appuyé
- `mousePosition` : position du curseur

InputSystem (nouveau système recommandé)

Système plus flexible et moderne pour gérer les entrées.

- Actions mappables
- Support multiplateforme amélioré
- Événements plutôt que polling

Les classes de gestion de temps

Time

Gère le temps écoulé et le framerate.

- `deltaTime` : temps depuis la dernière frame
- `fixedDeltaTime` : intervalle fixe pour FixedUpdate
- `timeScale` : multiplicateur de vitesse (pause si 0)
- `unscaledDeltaTime` : deltaTime non affecté par timeScale

Utilisation courante

Toujours multiplier les mouvements par `Time.deltaTime` pour une vitesse indépendante du framerate.

La classe Vector3

Fondamentale pour la 3D

Vector3 représente un point ou une direction dans l'espace 3D avec trois composantes (x, y, z).

Création et attributs

- `new Vector3(1f, 2f, 3f)` : création manuelle
- `Vector3.zero` : (0, 0, 0)
- `Vector3.one` : (1, 1, 1)
- `Vector3.forward` : direction (0, 0, 1)
- `v.magnitude` : longueur du vecteur
- `v.normalized` : vecteur unitaire dans la même direction

Opérations vectorielles essentielles

Opérations basiques

- Addition : `v1 + v2` — déplacement cumulé
- Soustraction : `v1 - v2` — direction et distance
- Multiplication scalaire : `v * 2f` — augmente la longueur
- `Vector3.Distance(v1, v2)` : distance entre deux points

Exemple pratique

Direction d'un objet vers une cible : `Vector3 direction = (target - transform.position).normalized;`

Produit scalaire et produit vectoriel

Produit scalaire `Vector3.Dot()`

Mesure l'alignement entre deux vecteurs.

- Résultat positif : vecteurs dans la même direction
- Résultat négatif : directions opposées
- Résultat zéro : vecteurs perpendiculaires

Produit vectoriel `Vector3.Cross()`

Crée un vecteur perpendiculaire aux deux vecteurs d'entrée.

- Utile pour calculer des rotations
- Pour déterminer des sens (gauche/droite)

Exemple

```
Vector3 right = Vector3.Cross(Vector3.up, forward);
```

Interpolation vectorielle

`Vector3.Lerp()`

Interpolation linéaire entre deux vecteurs.

Syntaxe

`Vector3.Lerp(start, end, t)` où `t` varie de 0 à 1

`Vector3.SmoothDamp()`

Interpolation progressive pour des mouvements fluides.

Utilisation courante

```
transform.position = Vector3.SmoothDamp(transform.position,
target, ref velocity, smoothTime);
// velocity est une variable de type Vector3 qui stocke la vitesse modifiée par la fonction.
// smoothTime contrôle la rapidité de l'interpolation.
```

Quaternions et rotations

Qu'est-ce qu'un Quaternion ?

Représentation mathématique d'une rotation en 3D sans risque de *gimbal lock*.

Utilisation pratique

- `Quaternion.Euler(x, y, z)` : crée depuis des angles
- `Quaternion.LookRotation(direction)` : rotation vers une direction
- `Quaternion.Lerp()` : interpolation lisse entre rotations
- `transform.rotation *= Quaternion.Euler(0, 45, 0)` : rotation incrémentale

Conseil

Préférez les Quaternions aux angles d'Euler pour éviter les problèmes d'interpolation.

Manipulation de la hiérarchie

Hiérarchie d'objets

- Les objets dans une scène sont organisés en une hiérarchie parent-enfant.
- Un objet enfant hérite des transformations de son parent (position, rotation, échelle).
- Vous pouvez faire glisser un objet dans la hiérarchie pour le rendre enfant d'un autre.

Fonctions de manipulation

- `transform.parent` : définit ou récupère le parent d'un objet.
- `transform.SetParent(Transform)` : change le parent d'un objet.
- `transform.GetChild(int)` : récupère un enfant par index.
- `transform.childCount` : nombre d'enfants d'un objet.

Manipulation des objets : création, recherche et destruction

Création d'objets

`GameObject.Instantiate(prefab, position, rotation)` : crée une instance d'un objet à partir d'un préfab.

Recherche d'objets

`GameObject.Find("Nom")` ou `GameObject.FindWithTag("Tag")` pour trouver des objets dans la scène.

Destruction d'objets

`GameObject.Destroy(objet)` : détruit un objet de la scène.

```
// Trouver un prefab dans le dossier Resources
GameObject prefab = Resources.Load<GameObject>("monprefab");
GameObject instance = Instantiate(prefab, position, rotation);
Destroy(instance, 5f); // détruit après 5 secondes
// ou Destroy(instance); pour détruire immédiatement
```

Manipulation des objets : diverses fonctions utiles

Activation/Désactivation d'objets

`gameObject.SetActive(b)` : active un objet (visible et interactif) ou non.

Déplacement d'objets

`transform.Translate(Vector3)` : translate dans une direction donnée.

Rotation d'objets

`transform.Rotate(Vector3)` : fait tourner autour de ses axes locaux.

Échelle d'objets

`transform.localScale` : modifie la taille d'un objet.

Parenté d'objets

`transform.SetParent(Transform)` : change le parent d'un objet, affectant sa position relative.

Manipulation des composants à un GameObject

Ajout de composants

`AddComponent<T>()` : ajoute un composant de type T à un objet.

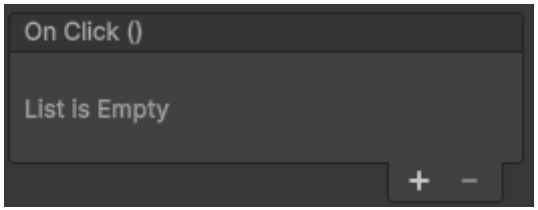
Récupération de composants

- `GetComponent<T>()` : récupère le composant de type T attaché.
- `GetComponents<T>()` : récupère tous les composants de type T.
- `GetComponentInChildren<T>()` : récupère le composant de type T dans les enfants d'un objet.
- `GetComponentInParent<T>()` : récupère le composant de type T dans les parents d'un objet.
- `TryGetComponent<T>(out T)` : tente de récupérer un composant de type T et retourne un booléen indiquant le succès.
- `GetComponent<T>(string)` : récupère un composant de type T avec un nom spécifique.
- `GetComponent<T>(Type)` : récupère un composant de type T à partir d'un type spécifié (utilisé pour les types non génériques).

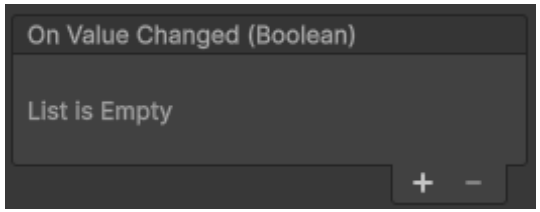
Gestion des événements et interactions

Système d'événements Unity

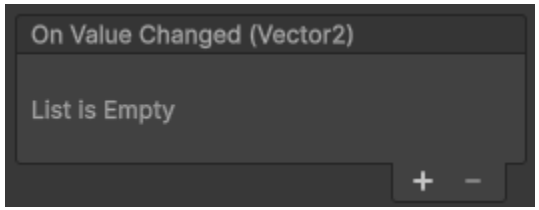
Dans l'éditeur vous avez des événements prédéfinis dont vous pouvez ajouter des fonctions à appeler lorsque ces événements sont déclenchés.



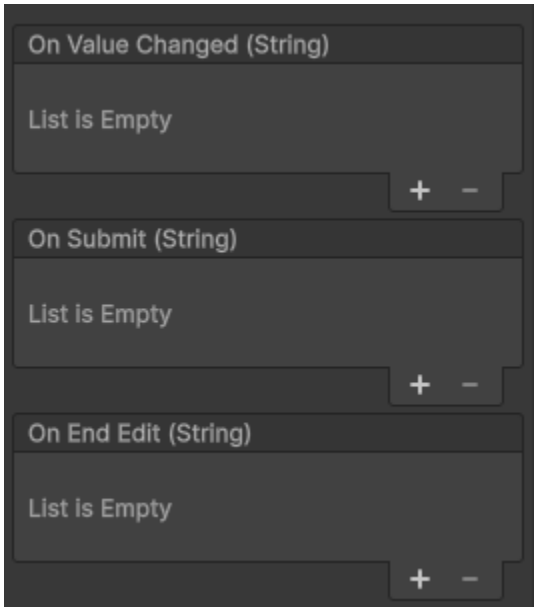
Button

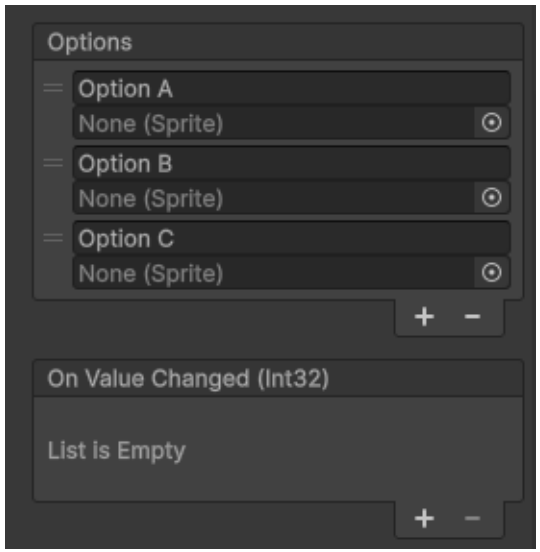


Toggle



ScrollView



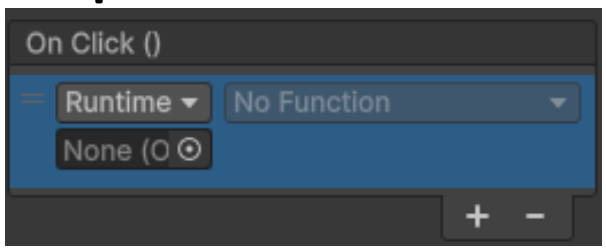


Ajout de fonctions à des événements

Processus d'ajout de fonctions

- Sélectionnez le composant avec l'événement.
- Dans l'inspecteur, trouvez la section de l'événement.
- Cliquez sur le "+" pour ajouter une nouvelle fonction.
- Faites glisser l'objet dans le champ "None (Object)".
- Sélectionnez la fonction à appeler dans la liste déroulante.

Étape 1

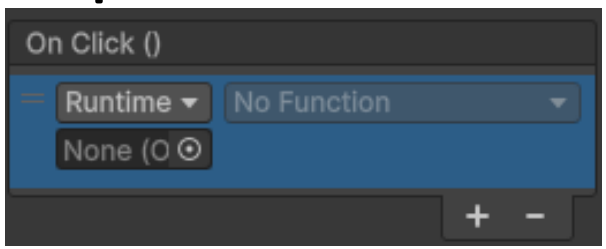


Ajout de fonctions à des événements

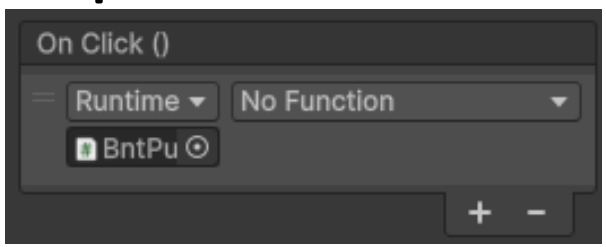
Processus d'ajout de fonctions

- Sélectionnez le composant avec l'événement.
- Dans l'inspecteur, trouvez la section de l'événement.
- Cliquez sur le "+" pour ajouter une nouvelle fonction.
- Faites glisser l'objet dans le champ "None (Object)".
- Sélectionnez la fonction à appeler dans la liste déroulante.

Étape 1



Étape 2

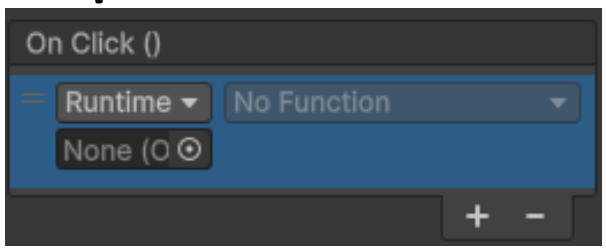


Ajout de fonctions à des événements

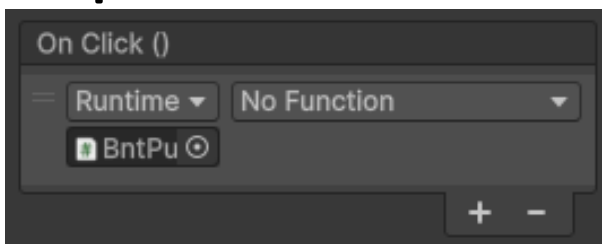
Processus d'ajout de fonctions

- Sélectionnez le composant avec l'événement.
- Dans l'inspecteur, trouvez la section de l'événement.
- Cliquez sur le "+" pour ajouter une nouvelle fonction.
- Faites glisser l'objet dans le champ "None (Object)".
- Sélectionnez la fonction à appeler dans la liste déroulante.

Étape 1



Étape 2



Étape 3